OpenFLUID
Software Environment
for Modelling Fluxes in Landscapes

# TP2 : Production de variables en sortie

| | |
|---|---|
| **Objectifs:** | Produire des variables sur différentes unités spatiales, utiliser des boucles de parcours de l'espace |
| **Pré-requis:** | TP1 |

A partir de la fonction de simulation "vide" créée lors du TP1 (`formation.signal.prod`), nous allons générer un signal de pluie sous la forme d'une variable produite sur toutes les unités de la classe SU. Ce signal est généré par une sinusoïde adaptée pour produire des valeurs de pluie.

## 1 Code source

Les modifications du code source sont à apporter dans le fichier `.cpp`.

### 1.1 Signature

Nous allons tout d'abord déclarer dans la signature que la fonction va produire une nouvelle variable nommée `water.atm-surf.H.rain`. Cette déclaration se fait à l'aide de l'instruction `DECLARE_PRODUCED_VAR`. L'instruction `DECLARE_PRODUCED_VAR` comporte 4 paramètres :

- le nom de la variable produite
- la classe d'unité sur laquelle est produite la variable
- la description de la variable
- l'unité (SI) de la variable

Une fois complétée, la signature devrait être similaire à :

```
BEGIN_SIGNATURE_HOOK
  DECLARE_SIGNATURE_ID("formation.signal.prod");
  DECLARE_SIGNATURE_NAME("");
  DECLARE_SIGNATURE_DESCRIPTION("");

  DECLARE_SIGNATURE_VERSION("1.0");
  DECLARE_SIGNATURE_SDKVERSION;
  DECLARE_SIGNATURE_STATUS(openfluid::base::EXPERIMENTAL);

  DECLARE_SIGNATURE_DOMAIN("");
```

```
  DECLARE_SIGNATURE_PROCESS("");
  DECLARE_SIGNATURE_METHOD("");
  DECLARE_SIGNATURE_AUTHORNAME("Chuck Norris");
  DECLARE_SIGNATURE_AUTHOREMAIL("norris@gmail.com");

  DECLARE_PRODUCED_VAR("water.atm-surf.H.rain","SU","rainfall height on SU","m");
END_SIGNATURE_HOOK
```

Après construction/installation de la fonction, il est possible de vérifier si la signature a bien été modifiée en exécutant la commande `openfluid-engine -u formation.signal.prod` ou `openfluid-engine -r` (pour l'ensemble des fonctions disponibles).

### 1.2 runStep()

La méthode `runStep()` est appelée à chaque pas de temps. Nous allons y ajouter le code de calcul du signal qui sera produit sur chaque unité de la classe SU au travers de la variable `water.atm-surf.H.rain`. Pour cela, nous allons utiliser une boucle spatiale sur les SU, et produire la variable à l'aide de l'instruction `OPENFLUID_AppendVariable`.
Une boucle spatiale est déclarée et identifiée au travers de l'instruction `DECLARE_UNITS_ORDERED_LOOP`, elle débute avec `BEGIN_UNITS_ORDERED_LOOP` et se termine avec `END_LOOP`.
Le générateur du signal proposé pour cet exercice est basé sur une fonction sinus, paramétrée par le numéro de pas de temps courant, et modifiée un coefficient :

Soit $V$ la valeur du signal, $t_i$ le numéro de pas de temps courant, $k$ un coefficient d'amplitude

$$V = \sin(\frac{t_i}{k} \times \frac{\pi}{180}) + 1$$

Si vous le souhaitez, vous pouvez intégrer votre propre équation de génération du signal.
Une fois complété, la méthode runStep() devrait être similaire à :

```
bool runStep(const openfluid::base::SimulationStatus* SimStatus)
{
  openfluid::core::Unit* pSU;  // pointeur sur la SU courante
  openfluid::core::ScalarValue Value;  // valeur du signal à calculer
  DECLARE_UNITS_ORDERED_LOOP(1);  // declaration d'une boucle spatiale d'identifiant 1

  BEGIN_UNITS_ORDERED_LOOP(1,"SU",pSU);  // debut de la boucle spatiale

    // calcul de la valeur du signal
    Value = (std::sin((SimStatus->getCurrentStep()/5)*3.1415926/180) + 1);

    // production de la valeur du signal
    OPENFLUID_AppendVariable(pSU,"water.atm-surf.H.rain",Value);

  END_LOOP;  // fin de la boucle spatiale

  return true;
}
```

## 2 Simulation

Pour la simulation, nous allons utiliser le jeu de données "Bassin versant virtuel". Ce jeu de données doit être déposé dans un répertoire qui sera ensuite utilisé par le moteur de calcul. (par

exemple /home/nomutilisateur/formationOF/dataset).
Le jeu de données comporte 24 unités de classe SU, 7 unités de classe RS, et est paramétré pour une simulation sur une période du 28 avril au 2 mai 1998 avec un pas de temps d'échange de 60 secondes.

## 2.1 Préparation du jeu de données

Afin que notre fonction de simulation soit prise en compte dans le modèle que l'on va exécuter, elle doit être déclarée dans le fichier `model.xml`. Pour celà, il faut modifier ce fichier et rajouter cette fonction entre les balises <model> et </model> comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<openfluid>
  <model>
    <function fileID="formation.signal.prod"/>
  </model>
</openfluid>
```

**Note:** En XML, les commentaires commencent par <!-- et se terminent par -->

## 2.2 Exécution

Pour lancer une simulation à partir du jeu de données, nous allons utiliser `openfluid-engine` en précisant le répertoire du jeu de données en entrée et celui des résultats.

- jeu de donnée en entrée : /home/nomutilisateur/formationOF/dataset
- résultats : /home/nomutilisateur/formationOF/outputs/TP2

La commande à exécuter est donc :

```
openfluid-engine -i /home/nomutilisateur/formationOF/dataset
  -o /home/nomutilisateur/formationOF/outputs/TP2
```

(à taper sur une seule ligne)

Si tout s'est bien passé, les résultats de la simulation sont accessibles dans /home/nomutilisateur/formationOF/outputs/TP2.